Ghost Riders: Sybil Attacks on Crowdsourced Mobile Mapping Services

Gang Wang, Bolun Wang, Tianyi Wang, Ana Nika, Haitao Zheng, *Fellow, IEEE*, and Ben Y. Zhao, *Senior Member, IEEE*

***Abstract*— Real-time crowdsourced maps, such as Waze provide timely updates on traffic, congestion, accidents, and points of interest. In this paper, we demonstrate how lack of strong location authentication allows creation of software-based *Sybil devices* that expose crowdsourced map systems to a variety of security and privacy attacks. Our experiments show that a single Sybil device with limited resources can cause havoc on Waze, reporting false congestion and accidents and automatically rerouting user traffic. More importantly, we describe techniques to generate Sybil devices at scale, creating armies of virtual vehicles capable of remotely tracking precise movements for large user populations while avoiding detection. To defend against Sybil devices, we propose a new approach based on *co-location edges*, authenticated records that attest to the one-time physical co-location of a pair of devices. Over time, co-location edges combine to form large *proximity graphs* that attest to physical interactions between devices, allowing scalable detection of virtual vehicles. We demonstrate the efficacy of this approach using large-scale simulations, and how they can be used to dramatically reduce the impact of the attacks. We have informed Waze/Google team of our research findings. Currently, we are in active collaboration with Waze team to improve the security and privacy of their system.**

***Index Terms*— Online social networks, crowdsourcing, Sybil attack, location privacy.**

I. Introduction

CROWDSOURCING is indispensable as a real-time data gathering tool for today's online services. Take for example map and navigation services. Both Google Maps and Waze use periodic GPS readings from mobile devices to infer traffic speed and congestion levels on streets and highways. Waze, the most popular crowdsourced map service, offers users more ways to actively share information on accidents, police cars, and even contribute content like editing roads, landmarks, and local fuel prices. This and the ability to interact with nearby users made Waze extremely popular, with an estimated 50 million users when it was acquired by Google for a reported $1.3 Billion USD in June 2013. Today, Google integrates selected crowdsourced data (*e.g.* accidents) from Waze into its own Maps application.

Unfortunately, systems that rely on crowdsourced data are inherently vulnerable to mischievous or malicious users seeking to disrupt or game the system [1]. For example, business owners can badmouth competitors by falsifying negative reviews on Yelp or TripAdvisor, and FourSquare users can forge their physical locations for discounts [2], [3]. For location-based services, these attacks are possible because there are no widely deployed tools to authenticate the location of mobile devices. In fact, there are few effective tools today to identify whether the origin of traffic requests are real mobile devices or software scripts.

The goal of our work is to explore the vulnerability of today's crowdsourced mobile apps against *Sybil devices*, software scripts that appear to application servers as "virtual mobile devices."[1] While a single Sybil device can damage mobile apps through misbehavior, larger groups of Sybil devices can overwhelm normal users and significantly disrupt any crowdsourced mobile app. In this paper, we identify techniques that allow malicious attackers to reliably create large populations of Sybil devices using software. Using the context of the Waze crowdsourced map service, we illustrate the powerful Sybil device attack, and then develop and evaluate robust defenses against them.

While our experiments and defenses are designed with Waze (and crowdsourced maps) in mind, our results generalize to a wide range of mobile apps. With minimal modifications, our techniques can be applied to services ranging from Foursquare and Yelp to Uber, YikYak and Pokemon Go, allowing attackers to cheaply emulate numerous virtual devices with forged locations to overwhelm these systems via misbehavior. Misbehavior can range from falsely obtaining coupons on Foursquare/Yelp, gaming the new user coupon system in Uber, imposing censorship on YikYak, to cheating in the game play of Pokemon Go. We believe our proposed defenses can be extended to these services as well. We discuss broader implications of our work in Section IX.

Sybil Attacks in Waze: In the context of Waze, our experiments reveal a number of potential attacks by Sybil devices. First is simple *event forgery*, where devices can generate fake events to the Waze server, including congestion,

Manuscript received September 30, 2016; revised March 27, 2017 and December 14, 2017; accepted March 4, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Guan. Date of publication April 12, 2018; date of current version June 14, 2018. This work was supported by the NSF under Grant CNS-1527939, Grant CNS-1224100, Grant CNS-1705042, and Grant CNS-1717028. *(Corresponding author: Gang Wang.)*

G. Wang is with the Department of Computer Science, Virginia Tech, Blacksburg, VA 24060 USA (e-mail: gangwang@vt.edu).

B. Wang is with the Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106 USA.

T. Wang is with ByteDance Inc., Beijing 100084, China.

A. Nika was with the Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106 USA. She is now with Microsoft Corporation, Redmond, WA 98052 USA.

H. Zheng and B. Y. Zhao are with the Department Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106 USA, and also with the Department of Computer Science, The University of Chicago, Chicago, IL 60637 USA.

Digital Object Identifier 10.1109/TNET.2018.2818073

[1]We refer to these scripts as Sybil devices, since they are the manifestations of Sybil attacks [4] in the context of mobile networks.

accidents or police activity that might affect user routes. Second, we describe techniques to reverse engineer mobile app APIs, thus allowing attackers to create lightweight scripts that effectively emulate a large number of virtual vehicles that collude under the control of a single attacker. We call Sybil devices in Waze "ghost riders." These Sybils can effectively magnify the efficacy of any attack, and overwhelm contributions from any legitimate users. Finally, we discover a significant privacy attack where ghost riders can silently and invisibly "follow" and precisely track individual Waze users throughout their day, precisely mapping out their movement to work, stores, hotels, gas station, and home. We experimentally confirmed the accuracy of this attack against our own vehicles, quantifying the accuracy of the attack against GPS coordinates. Magnified by an army of ghost riders, an attacker can potentially track the constant whereabouts of millions of users, all without any risk of detection.

Defenses: Prior proposals to address the location authentication problem have limited appeal, because of reliance on widespread deployment of specialized hardware, either as part of physical infrastructure, *i.e.*, cellular base stations, or as modifications to mobile devices themselves. Instead, we propose a practical solution that limits the ability of Sybil devices to amplify the potential damage incurred by any single attacker. We introduce *collocation edges*, authenticated records that attest to the one-time physical proximity of a pair of mobile devices. The creation of collocation edges can be triggered opportunistically by the mapping service, *e.g.*, Waze. Over time, collocation edges combine to form large *proximity graphs*, network structures that attest to physical interactions between devices. Since ghost riders cannot physically interact with real devices, they cannot form direct edges with real devices, only indirectly through a small number of real devices operated by the attacker. Thus, the edges between an attacker and the rest of the network are limited by the number of real physical devices she has, regardless of how many ghost riders are under her control. This reduces the problem of detecting ghost riders to a community detection problem on the proximity graph (The graph is seeded by a small number of trusted infrastructure locations).

Our paper includes these key contributions:

- We explore limits and impacts of single device attacks on Waze, *e.g.*, artificial congestion and events.
- We describe techniques to create light-weight ghost riders, virtual vehicles emulated by client-side scripts, through reverse engineering of the Waze app's communication protocol with the server.
- We identify a new privacy attack that allows ghost riders to virtually follow and track individual Waze users in real-time, and describe techniques to produce precise, robust location updates.
- We propose and evaluate defenses against ghost riders, using *proximity graphs* constructed with edges representing authenticated collocation events between pairs of devices. Since collocation can only occur between pairs of physical devices, proximity graphs limit the number of edges between real devices and ghost riders, thus isolating groups of ghost riders and making them detectable using community detection algorithms.

Impacts: We have informed the Google/Waze team of our findings, and our efforts have led to significant improvements to the security and privacy of Waze system. In addition to Waze and Google Maps, there is more and more evidence of real-world Sybil threats in similar services, including "ghost drivers" in Uber who generate fake rides to earn financial bonus [5], [6], and cheaters in Pokemon Go by spoofing GPS data [7], [8]. Our study provides insights and mechanisms to counter such attacks.

II. Waze Background

Waze is the most popular crowdsourced navigation app on smartphones, with more than 50 million users when it was acquired by Google in June 2013 [9]. Waze collects GPS values of users' devices to estimate real-time traffic. It also allows users to report on-road events such as accidents, road closures and police vehicles, as well as editing roads and even updating local fuel prices. Some features, *e.g.*, user reported accidents, have been integrated into Google Maps [10]. Here, we briefly describe the key functionality in Waze as context for our work.

Trip Navigation: Waze's main feature is assist users to find the best route to their destination and turn-by-turn navigation. Waze generates aggregated real-time traffic updates using GPS data from its users, and optimizes user routes both during trip planning and during navigation. If and when traffic congestions is detected, Waze automatically re-routes users towards an alternative.

Crowdsourced User Reports: Waze users can generate real-time *event reports* on their routes to inform others about ongoing incidents. Events range from accidents to road closures, hazards, and even police speed traps. Each report can include a short note with a photo. The event shows up on the map of users driving towards the reported location. As users get close, Waze pops up a window to let the user "say thanks," or report the event is "not there." If multiple users choose "not there", the event will be removed. Waze also merges multiple reports of the same event type at the same location into a single event.

Social Function: To increase user engagement, Waze supports simple social interactions. Users can see avatars and locations of nearby users. Clicking on a user's avatar shows more detailed user information, including nickname, ranking, and traveling speed. Also, users can send messages and chat with nearby users. This social function gives users the sense of a large community. Users can elevate their rankings in the community by contributing and receiving "thanks" from others.

III. Attacking Crowdsourced Maps

In this section, we describe basic attacks to manipulate Waze by generating false road events and fake traffic congestion. Since Waze relies on real-time data for trip planning and route selection, these attacks can influence user's routing decisions. Attackers can attack specific users by forging congestion to force automatic rerouting on their trips. The attack is possible because Waze has no reliable authentication on user reported data, such as their GPS.

We first discuss experimental ethics and steps we took to limit impact on real users. Then, we describe basic mechanisms and resources needed to launch attacks, and use

controlled experiments on two attacks to understand their feasibility and limits. One attack creates fake road events at arbitrary locations, and the other seeks to generate artificial traffic hotspots to influence user routing.

A. Ethics

Our experiments seek to understand the feasibility and limits of practical attacks on crowdsourcing maps like Waze. We are very aware of the potential impact to real users from any experiments. We consulted our local IRB and have taken all possible precautions to ensure that our experiments do not negatively impact real Waze users. In particular, we choose experiment locations where user population density is extremely low (unoccupied roads), and only perform experiments at low-traffic hours, *e.g.*, between 2am and 5am. During experiments, we continuously scan the entire experiment region and neighboring areas, to ensure no other Waze users (except our own accounts) are within miles of the test area. If any Waze users are detected, we immediately terminate all running experiments. Our study received the IRB approval under protocol# COMS-ZH-YA-010-7N.

Our work is further motivated by our view of the risks of inaction versus risks posed to users by our study. On one hand, we can and have minimized risk to Waze users during our study, and we believe our experiments have not affected any Waze users. On the other hand, we believe the risk to millions of Waze users from pervasive location tracking (Section V) is realistic and potentially very damaging. We feel that investigating these attacks and identifying these risks to the broad community at large was the ethically correct course of action. Furthermore, full understanding of the attacks was necessary to design a *practical* defense.

B. Basic Attack: Generating Fake Events

Launching attacks against crowdsourced maps like Waze requires three steps: automate input to mobile devices that run the Waze app; control the device GPS and simulate device movements (*e.g.*, car driving); obtain access to *multiple* devices. All three are easily achieved using widely available mobile device emulators.

Most mobile emulators run a full OS (*e.g.*, Android, iOS) down to the kernel level, and simulate hardware features such as camera, SDCard and GPS. We choose the GenyMotion Android emulator [11] for its performance and reliability. Attackers can automatically control the GenyMotion emulator via Monkeyrunner scripts [12]. They can generate user actions such as clicking buttons and typing text, and feed pre-designed GPS sequences to the emulator (through a command line interface) to simulate location positioning and device movement. By controlling the timing of the GPS updates, they can simulate any "movement speed" of the simulated devices.

Using these tools, attackers can generate fake events (or alerts) at a given location by setting fake GPS on their virtual devices. This includes any events supported by Waze, including accidents, police, hazards, and road closures. We find that a single emulator can generate any event at arbitrary locations on the map. We validate this using experiments on a variety of unoccupied roads, including highways, local and rural roads (50+ locations, 3 repeated tests each). Note that our experiments only involve data in the Waze system, and do not affect real road vehicles not running the Waze app. Thus "unoccupied" means no vehicles on the road with mobile devices actively running the Waze app. After creation, the fake event stays on the map for about 30 minutes. Any Waze user can report that an event was "not there." We find it takes two consecutive "not theres" (without any "thanks" in between) to delete the event. Thus an attacker can ensure an event persists by occasionally "driving" other virtual devices to the region and "thanking" the original attacker for the event report.

Fig. 1. Before the attack (left), Waze shows the fastest route for the user. After the attack (right), the user gets automatically re-routed by the fake traffic jam.

C. Congestion and Traffic Routing

A more serious attack targets Waze's real-time trip routing function. Since route selection in Waze relies on predicted trip time, attackers can influence routes by creating "fake" traffic hotspots at specific locations. This can be done by configuring a group of virtual vehicles to travel slowly on a chosen road segment.

We use controlled experiments to answer two questions. First, under what conditions can attackers successfully create traffic hotspots? Second, how long can an artificial traffic hotspot last? We select three low-traffic roads in the state of Texas that are representative of three popular road types based on their speed limit—Highway (65 mph), Local (45 mph) and Residential (25 mph). To avoid real users, we choose roads in low population rural areas, and run tests at hours with the lowest traffic volumes (usually 3-5AM). We constantly scan for real users in or nearby the experimental region, and reset/terminate experiments if users come close to an area with ongoing experiments. Across all our experiments, only 2 tests were terminated due to detected presence of real users nearby. Finally, we have examined different road types and hours of the day to ensure they do not introduce bias into our results.

Creating Traffic Hotspots: Our experiment shows that it only takes one slow moving car to create a traffic congestion, when there are no real Waze users around.

Waze displays a red overlay on the road to indicate traffic congestion (Figure 1, right). Different road types have different congestion thresholds, with thresholds strongly correlated to the speed limit. The congestion thresholds for Highway, Local and Residential roads are 40mph, 20mph and 15mph, respectively.

To understand if this is generalizable, we repeat our tests on other unoccupied roads in different states and countries.

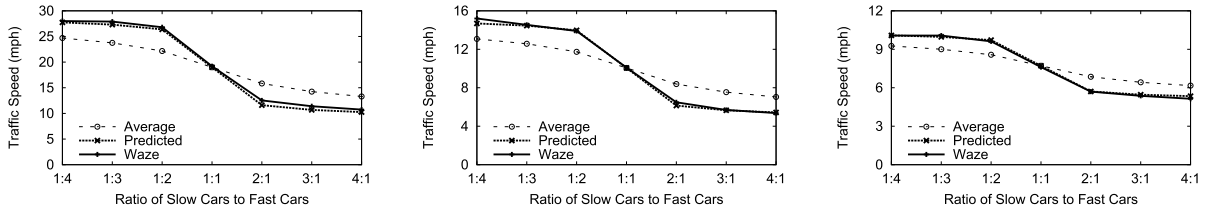


Fig. 2. The traffic speed of the road with respect to different combinations of number of slow cars and fast cars. We show that Waze is not using the average speed of all cars, and our inferred function can correctly predict the traffic speed displayed on Waze. (a) Highway. (b) Local Road. (c) Residential.

We picked 18 roads in five states in the US (CO, MO, NM, UT, MS) and British Columbia, Canada. In each region, we select three roads with different speed limits (highway, local and residential). We find consistent results: a single virtual vehicle can always generate a traffic hotspot; and the congestion thresholds were consistent across different roads of the same speed limit.

Outvoting Real Users: Generating traffic hotspot in practical scenarios faces a challenge from real Waze users who drive at normal (non-congested) speeds: attacker's virtual vehicles must "convince" the server there's a stream of slow speed traffic on the road even as real users tell the server otherwise. We need to understand how Waze aggregated multiple inputs to estimate traffic speed.

We perform an experiment to infer this aggregation function used by Waze. We create two groups of virtual vehicles: N_s slow-driving cars with speed S_s, and N_f fast-driving cars with speed S_f; and they all pass the target location at the same time. We study the congestion reported by Waze to infer the aggregation function. Note that the server-estimated traffic speed is visible on the map *only if* we formed a traffic hotspot. We achieve this by setting the speed tuple (S_s, S_f) to (10mph, 30mph) for Highway, (5, 15) for Local and (5, 10) for Residential.

As shown in Figure 2, when we vary the ratio of slow cars over fast cars (N_s:N_f), the Waze server produces different final traffic speeds. We observe that Waze does not simply compute an "average" speed over all the cars. Instead, it uses a weighted average with higher weight on the majority cars' speed. We infer an aggregation function as follows:

$$S_{waze} = \frac{S_{max} \cdot max(N_s, N_f) + S_{avg} \cdot min(N_s, N_f)}{N_s + N_f}$$

where $S_{avg} = \frac{S_s N_s + S_f N_f}{N_s + N_f}$, and S_{max} is the speed of the group with N_{max} cars. As shown in Figure 2, our function can predict Waze's aggregate traffic speed accurately, for all different types of roads in our test. For validation purposes, we run another set of experiments by raising S_f above the hotspot thresholds (65mph, 30mph and 20mph respectively for the three roads). We can still form traffic hotspots by using more slow-driving cars ($N_s > N_f$), and our function can still predict the traffic speed on Waze accurately.

Long-Lasting Traffic Congestion: A traffic hotspot will last for 25-30 minutes if no other cars drive by. Once aggregate speed normalizes, the congestion event is dismissed within 2-5 minutes. To create a long-lasting virtual traffic jam, attackers can simply keep sending slow-driving cars to the congestion area to resist the input from real users. We validate this using a simple, 50-minute long experiment where 3 virtual vehicles create a persistent congestion by driving slowly through an area, and then looping back every 10 minutes. Meanwhile, 2 other virtual cars emulate legitimate drivers that pass by at high speed every 10 minutes. We find the traffic hotspot persists for the entire experiment period.

Impact on End Users: Waze uses real-time traffic data to optimize routes during trip planning. Waze estimates the end-to-end trip time and recommends the fastest route. Once on the road, Waze continuously estimates the travel time, and automatically reroutes if the current route becomes congested. An attacker can launch physical attacks by placing fake traffic hotspots on the user's original route. While congestion alone does not trigger rerouting, Waze reroutes the user to a detour when the estimated travel time through the detour is shorter than the current congested route (see Figure 1).

We also note that Waze data is used by Google Maps, and therefore can potentially impact their 1+ billion users [13]. Our experiment shows that artificial congestion do not appear on Google Maps, but fake events generated on Waze are displayed on Google Maps without verification, including "accidents", "construction" and "objects on road". Finally, event updates are synchronized on both services, with a 2-minute delay and persist for a similar period of time (*e.g.*, 30 minutes).

IV. Sybil Attacks

So far, we have shown that attackers using emulators can create "virtual vehicles" that manipulate the Waze map. An attacker can generate much higher impact using a large group of virtual vehicles (or *Sybils* [4]) under control. In this section, we describe techniques to produce light-weight virtual vehicles in Waze, and explore the scalability of the group-based attacks. We refer to large groups of virtual vehicles as "ghost riders" for two reasons. First, they are easy to create en masse, and can travel in packs to outvote real users to generate more complex events, *e.g.*, persistent traffic congestion. Second, as we show in §V, they can make themselves invisible to nearby vehicles.

A. Creating Sybil Devices

We start by looking at the limits of the large-scale Sybil attacks on Waze. First, we note user accounts do not pose a challenge to attackers, since account registration can be fully automated. We found that a single-threaded Monkeyrunner script could automatically register 1000 new accounts in a day.

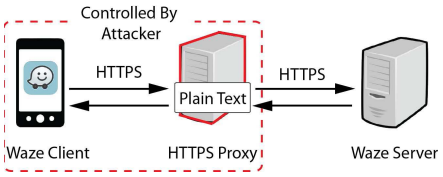


Fig. 3. Using a HTTPS proxy as man-in-the-middle to intercept traffic between Waze client and server.

The limiting factor is the scalability of vehicle emulation. Even though emulators like GenyMotion are relatively lightweight, each instance still takes significant computational resources. For example, a MacBookPro with 8G of RAM supports only 10 simultaneous emulator instances. For this, we explore a more scalable approach to client emulation that can increase the number of supported virtual vehicles by orders of magnitude. Specifically, we reverse engineer the communication APIs used by the app, and replace emulators with simple Python scripts that mimic API calls.

Reverse Engineering Waze APIs: The Waze app uses HTTPS to communicate with the server, so API details cannot be directly observed by capturing network traffic (TLS/SSL encrypted). However, an attacker can still intercept HTTPS traffic, by setting up a proxy [14] between her phone and Waze server as a man-in-the-middle attack [15], [16]. As shown in Figure 3, an attacker needs to pre-install the proxy server's root Certificate Authorities (CA) to her own phone as a "trusted CA." This allows the proxy to present self-signed certificates to the phone claiming to be the Waze server. The Waze app on the phone will trust the proxy (since the certificate is signed by a "trusted CA"), and establish HTTPS connections with the proxy using proxy's public key. On the proxy side, the attacker can decrypt the traffic using proxy's private key, and then forward traffic from the phone to Waze server through a separate TLS/SSL channel. The proxy then observes traffic to the Waze servers and extracts the API calls from plain text traffic.

Hiding API calls using traffic encryption is fundamentally challenging, because the attacker has control over most of the components in the communication process, including phone, the app binary, and the proxy. A known countermeasure is certificate pinning [17], which embeds a copy of the server certificate within the app. When the app makes HTTPS requests, it validates the server-provided certificate with its known copy before establishing connections. However, dedicated attackers can extract and replace the embedded certificate by disassembling the app binary or attaching the app to a debugger [18], [19].

Once we obtain the knowledge of Waze APIs, we can build extremely lightweight Waze clients using python scripts, allocating one thread for each client. Within each thread, we login to the app using a separate account, and maintain a live session by sending periodic GPS coordinates to the Waze server.

B. Potential Defenses Against Sybil Devices

While attackers can easily create lightweight Sybil devices, it is nontrivial for services providers to effectively detect and defend against them. Below we discuss possible ways to reliably authenticate mobile devices, and highlight the key challenges to do so.

Email Verification: A straight-forward approach is to authenticate a mobile device via an email account. However, attackers may create fake email accounts automatically or purchase them in bulks from blackmarkets [20]. This approach has limited effect.

SMS Verification: Two-factor Authentication can be used to verify phone numbers. The latest Waze app already requires SMS verification during account registration. However, attackers can bypass this using disposable phone numbers or temporal SMS services [21].

CAPTCHA: Service providers can use CAPTCHAs to test whether a phone is operated by a human user or a computer script. This approach has key limitations too. First, solving CAPTCHAs on smartphones can be distracting and annoying to legitimate users. Second, attackers can leverage crowdsourced CAPTCHA farms to solve CAPTCHAs in real time [22].

IMEI Validation: Service providers may also consider validating the unique identifier of the phone such as IMEI. But the challenge is there are already public IMEI databases [23] or fake IMEI generators [24] that can help attackers to spoof the identifier.

Device Fingerprinting: Researchers have proposed to use motion sensors to fingerprint smartphones [25]. The idea is that smartphone sensors such as accelerometers and gyroscopes usually have anomalies in their signals due to manufacturing imperfections. Such signal anomalies can be used to uniquely fingerprint the phone. However, a more recent result shows that fingerprinting accuracy would drop quickly for a large number of devices (*e.g.*, 100K) [26]. This technique is still not reliable enough to authenticate mobile devices.

IP Verification: Finally, service providers can also check if the device's IP is an actual mobile IP (or a suspicious web proxy). However, attacker can overcome this by routing their traffic through a cellular data plan.

We find that authenticating individual mobile devices is very challenging. As long as attackers have full controls on the client side, they could (easily) forge the data needed for authentication. In the later section (§VI), we will describe our method to detect groups of Sybil devices.

C. Scalability of Ghost Riders

Ghost riders are fully functional Waze clients and they are highly scalable. Each ghost rider is scripted not only to report GPS to Waze server, but also report fake events using the API. We run 1000 virtual vehicles on a single Linux Dell Server (Quad Core, 2GB RAM), and find that at steady state, 1000 virtual devices only introduces a small overhead: 11% of memory usage, 2% of CPU and 420 Kbps bandwidth. In practice, attackers can easily run tens of thousands of virtual devices on a commodity server.

Finally, we experimentally confirm the practical efficacy and scalability of ghost riders. We chose a secluded highway in rural Texas, and used 1000 virtual vehicles (hosted on a single server and single IP) to generate a highly congested traffic

hotspot. We perform our experiment in the middle of the night after repeated scans showed no Waze users within miles of our test area. We positioned 1000 ghost riders one after another, and drove them slowly at 15 mph along the highway, looping them back every 15 minutes for an entire hour. The congestion shows up on Waze 5 minutes after our test began, and stayed on the map during the entire test period. No problems were observed during our test, and tests to generate fake events (accidents etc.) also succeeded.

V. User Tracking Attack

Next, we describe a powerful new attack on user privacy, where virtual vehicles can track Waze users continuously without risking detection themselves. By exploiting a key social functionality in Waze, attackers can remotely follow (or stalk) any individual user in real time. This is possible with single device emulation, but greatly amplified with the help of large groups of ghost riders, possibly tracking large user populations simultaneously and putting user (location) privacy at great risk. We start by examining the feasibility (and key enablers) of this attack. We then present a simple but highly effective tracking algorithm that follows individual users in real time, which we have validated using real life experiments (with ourselves as the targets).

The only way for Waze users to avoid tracking is to go "invisible" in Waze. However, doing so forfeits the ability to generate reports or message other users. Waze also resets the invisible setting every time the app is opened [27].

A. Feasibility of User Tracking

A key feature in Waze allows users to socialize with others on the road. Each user sees on her screen icons representing the locations of nearby users, and can chat or message with them through the app. Leveraging this feature, an attacker can pinpoint any target who has the Waze app running on her phone. By constantly "refreshing" the app screen (issuing an update query to the server), an attacker can query the victim's GPS location from Waze in real time. To understand this capability, we perform detailed measurements on Waze to evaluate the efficiency and precision of user tracking.

Tracking via User Queries: A Waze client periodically requests updates in her nearby area, by issuing an update query with its GPS coordinates and a rectangular "search area." This search area can be set to any location on the map, and does not depend on the requester's own location. The server returns a list of users located in the area, including userID, nickname, account creation time, GPS coordinates and the GPS timestamp. Thus an attacker can find and "follow" a target user by first locating them at any given location (work, home) and then continuously following them by issuing update queries centered on the target vehicle location, all automated by scripts.

Overcoming Downsampling: The user query approach faces a downsampling challenge, because Waze responds to each query with an "incomplete" set of users, *i.e.*, up to 20 users per query regardless of the search area size. This downsampled result is necessary to prevent flooding the app screen with too many user icons, but it also limits an attacker's ability to follow a moving target. We find that this downsampling can be overcome by simply repeatedly querying the system until the target is found. We perform query measurements on four test areas (of different sizes between 3×4 mile^2 and 24×32 mile^2) in the downtown area of Los Angeles (City A, with 10 million residents as of 2015). For each area, we issue 400 queries within 10 seconds, and examine the number of unique users returned by all the queries. Results in Figure 4 show that the number of unique users reported converges after 150-250 queries for the three small search areas ($\leq 12 \times 16$ mile^2). For the area of size 24×32 mile^2, more than 400 queries are required to reach convergence.

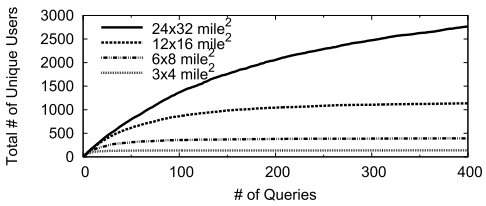


Fig. 4. # of queries vs. unique returned users in the area.

Tracking Users Over Time: Our analysis found that each active Waze app updates its GPS coordinates to the server every 2 minutes, regardless of whether the user is mobile or stationary. Even when running in the background, the Waze app reports GPS values every 5 minutes. As long as the Waze app is open (even running in the background), the user's location is continuously reported to Waze and potential attackers. Clearly, a more conservative approach to managing location data would be helpful here.

We note that attackers can perform long-term tracking on a target user (*e.g.*, over months). The attacker needs a persistent ID associated to the target. The "userID" field in the metadata is insufficient, because it is a random "session" ID assigned upon user login and is released when the user kills the app. However, the "account creation time" can serve as a persistent ID, because a) it remains the same across the user's different login sessions, and b) it is precise down to the second, and is sufficiently to uniquely identify single users in the same geographic area. While Waze can remove the "account creation time" field from metadata, a persistent attacker can overcome this by analyzing the victim's mobility pattern. For example, the attacker can identify a set of locations where the victim has visited frequently or stayed during the past session, mapping to home or workplace. Then the attacker can assign a ghost rider to constantly monitor those areas, and re-identify the target once her icon shows up in a monitored location, *e.g.*, home.

Stealth Mode: We note that attackers remain invisible to their targets, because queries on any specific geographic area can be done by Sybils operating "remotely," i.e. claiming to be in a different city, state or country. Attackers can enable their "invisible" option to hide from other nearby users. Finally, disabling these features still does not make the attacker visible. Waze only updates each user's "nearby" screen every 2 minutes (while sending its own GPS update to the servers). Thus a tracker can "pop into" the target's region, query for the target, and then move out of the target's observable range, all before the target can update and detect it.

TABLE I
TRACKING EXPERIMENT RESULTS

Location	Route Length (Mile)	Travel Time (Minute)	GPS Sent By Victim	GPS Captured by Attacker	Followed to Destination?	Avg. Track Delay (Second)	Waze User Density (# of Users / mile^2)
City A	12.8	35	18	16	Yes	43.79	56.6
Highway B	36.6	40	20	19	Yes	9.24	2.8

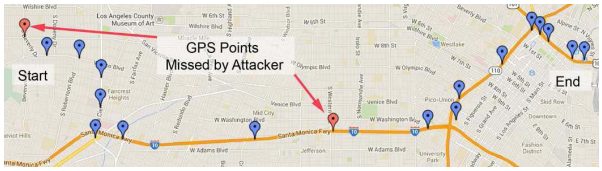


Fig. 5. A graphical view of the tracking result in Los Angeles downtown (City A). Blue dots are GPS points captured by the attacker and the red dots are those missed by the attacker.

B. Real-Time Individual User Tracking

To build a detailed trace of a target user's movements, an attacker first bootstraps by identifying the target's icon on the map. This can be done by identifying the target's icon while confirming her physical presence at a time and location. The attacker centers its search area on the victim's location, and issues a large number of queries (using Sybil accounts) until it captures the next GPS report from the target. If the target is moving, the attacker moves the search area along the target's direction of movement and repeats the process to get updates.

Experiments: To evaluate its effectiveness, we performed experiments by tracking one of our own Android smartphones and one of our virtual devices. Tracking was effective in both cases, but we experimented more with tracking our virtual device, since we could have it travel to any location. Using the OSRM tool [28], we generate detailed GPS traces of two driving trips, one in downtown area of Los Angeles (City A), and one along the interstate highway-101 (Highway B). The target device uses a realistic driving speed based on average traffic speeds estimated by Google Maps during the experiment. The attacker used 20 virtual devices to query Waze simultaneously in a rectangular search area of size 6×8 mile^2. This should be sufficient to track the GPS update of a fast-driving car (up to 160 mph). Both experiments were during morning hours, and we logged both the network traffic of the target phone and query data retrieved by the attacker. Note that we did not generate any "events" or otherwise affect the Waze system in this experiment.

Results: Table I lists the results of tracking our virtual device, and Figure 5 presents a graphical view of the City A result. For both routes, the attacker can consistently follow the victim to her destination, though the attacker fails to capture 1-2 GPS points out of the 18-20 reported. For City A, the tracking delay, *i.e.*, the time spent to capture the subsequent GPS of the victim, is larger (averaging 43s rather than 9s). This is because the downtown area has a higher Waze user density, and required more rounds of queries to locate the target.

Our experiments represent two highly challenging (*i.e.*, worst case) scenarios for the attacker. The high density of Waze users in City A downtown is makes it challenging to locate a target in real time with downsampling. On Highway B, the target travels at a high speed (∼60mph), putting a stringent time limit on the tracking latency, *i.e.*, the attacker must capture the target before he leaves the search area. The success of both experiments confirms the effectiveness and practicality of the proposed attack.

VI. DEFENSES

In this section, we propose defense mechanisms to significantly limit the magnitude and impact of these attacks. While individual devices can inflict limited damage, an attacker's ability to control a large number of virtual vehicles at low cost elevates the severity of the attack in both quantity and quality. Our priority, then, is to restrict the number of ghost riders available to each attacker, thus increasing the cost per "vehicle" and reducing potential damage.

The most intuitive approach is perform strong location authentication, so that attackers must use real devices physically located at the actual locations reported. This would make ghost riders as expensive to operate as real devices. Unfortunately, existing methods for location authentication do not extend well to our context. Some proposals solely rely on trusted infrastructures (*e.g.*, wireless access points) to verify the physical presence of devices in close proximity [29], [30]. However, this requires large scale retrofitting of cellular celltowers or installation of new hardware, neither of which is practical at large geographic scales. Others propose to embed tamperproof location hardware on mobile devices [31], [32], which incurs high cost per user, and is only effective if enforced across all devices. For our purposes, we need a scalable approach that works with current hardware, without incurring costs on mobile users or the map service (Waze).

A. Sybil Detection via Proximity Graph

Instead of optimizing per-device location authentication, our proposed defense is a Sybil detection mechanism based on the novel concept of *proximity graph*. Specifically, we leverage physical proximity between real devices to create *collocation edges*, which act as secure attestations of shared physical presence. In a proximity graph, nodes are Waze devices (uniquely identified by an account username and password on the server side). They perform secure peer-to-peer location authentication with the Waze app running in the background. An edge is established if the proximity authentication is successful.

Because Sybil devices are scripted software, they are highly unlikely to come into physical proximity with real devices. A Sybil device can only form collocation edges with other Sybil devices (with coordination by the attacker) or the attacker's own physical devices. The resulting graph should have only very few (or no) edges between virtual devices and real users (other than the attacker). Leveraging prior work

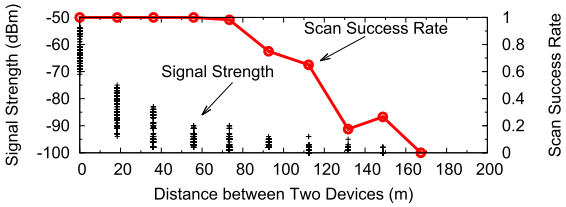


Fig. 6. WiFi signal strength and scan success rate with respect to car distance in static scenarios.

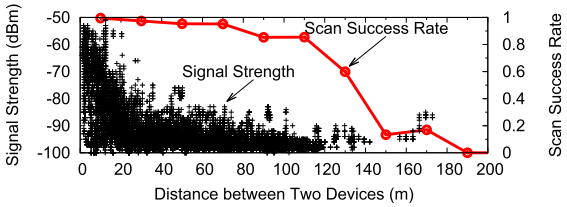


Fig. 7. WiFi signal strength and scan success rate with respect to car distance in driving scenarios.

on Sybil detection in social networks, groups of Sybils can be characterized by the few "attack edges" connecting them to the rest of the graph, making them identifiable through community-detection algorithms [33].

We use *a very small number* of trusted nodes only to bootstrap trust in the graph. We assume a small number of infrastructure access points are known to Waze servers, *e.g.*, hotels and public WiFi networks associated with physical locations stored in IP-location databases (used for geolocation by Apple and Google). Any Waze device that communicates with the Waze server under their IPs (and reports a GPS location consistent with the IP) automatically creates a new collocation edge to the trusted node.

B. Peer-Based Proximity Authentication

To build the proximity graph, we first need a reliable method to verify the *physical* collocation of mobile devices. We cannot rely on GPS reports since attackers can forge arbitrary GPS coordinates, or Bluetooth based device ranging [34] because the coverage is too short (<10 meters) for vehicles. Instead, we consider a challenge-based proximity authentication method, which leverages the limited transmission range of WiFi radios.

WiFi Tethering Challenge: We use the smartphone's WiFi radio to implement a proximity challenge between two Waze devices. Because WiFi radios have limited ranges (<250 meters for 802.11n [35])), two Waze devices must be in physical proximity to complete the challenge. Specifically, we (or the Waze server) instruct one device to enable WiFi tethering and broadcast beacons with an SSID provided by the Waze server, *i.e.*, a randomly generated, time-varying bit string. This bit string cannot be forged by other users or used to re-identify a particular user. The second device proves its proximity to the first device by returning the SSID value heard over the air to the Waze server.

The key concerns of this approach are whether the WiFi link between two vehicles is stable/strong enough to complete the challenge, and whether the separation distance is long enough for our needs. This concern is valid given the high moving speed, potential signal blockage from vehicles' metal components, and the low transmit power of smartphones. We explore these issues with detailed measurements on real mobile devices.

First, we perform measurements on stationary vehicles to study the joint effect of blockage and limited mobile transmit power. We put two Android phones into two cars (with windows and doors closed), one running WiFi tethering to broadcast beacons and the other scanning for beacons. Figure 6 plots the WiFi beacon strength at different separation distances. We see that the above artifacts make the signal strength drop to –100 dBm before the distance reaches 250 meters. In the same figure, we also plot the probability of successful beacon decoding (thus challenge completion) across 400 attempts within 2 minutes. It remains 100% when the two cars are separated by <80 meters, and drops to zero at 160 meters.

Next, we perform driving experiments on a highway at normal traffic hours in the presence of other vehicles. The vehicles travel at speeds averaging 65 mph. During driving, we are able to vary the distance between the two cars, and use recorded GPS logs to calculate the separation distance. Figure 7 shows that while WiFi signal strength fluctuates during our experiments, the probability of beacon decoding remains very high at 98% when the separation is less than 80 meters but drops to <10% once the two cars are more than 140 meters apart.

Overall, the results suggest the proposed WiFi tethering challenge is a reliable method for proximity authentication for our system. In practice, Waze can start the challenge when detecting the two vehicles are within the effective range, *e.g.*, 80 meters. Since the WiFi channel scan is fast, *e.g.*, 1-2 seconds to do a full channel scan in our experiments, this challenge can be accomplished quickly with minimum energy cost on mobile devices.

Constructing Proximity Graphs: In a proximity graph, each node is a Waze device, and an edge indicates the two users come into physical proximity, *e.g.*, 80 meters, within a predefined time window. The resulting graph is undirected but weighted based on the number of times the two users have encountered. Using weighted graph makes it harder for Sybils to blend into the normal user region. Intuitively, real users will get more weights on their edges as they use Waze over time. For attackers, in order to blend in the graph, they need to build more weighted attack edges to real users (higher costs).

This approach should not introduce much energy consumption to users' phones. First, Waze server does not need to trigger collocation authentication every time two users are in close proximity. Instead, the proximity graph will be built up over time. A user only need to authenticate with other users occasionally, since we can require that device authentication expires after a moderate time period (*e.g.,* months) to reduce the net impact on wireless performance and energy usage. Second, since the process is triggered by the Waze server, Waze can can use WiFi sensing from devices to find "opportunistic" authentication times that minimize impact on performance and energy. Waze can also use one tether to simultaneously authenticate multiple colocated devices within an area. This further reduces authentication overhead, and avoids performance issues like wireless interference in areas with high user density. In practice, there might be users who

never turn on the Wifi permission for Waze. One possible strategy is to lower these users' weights in traffic aggregation and temporarily strict them from querying nearby users, to control the potential damage. Also, WiFi is just one example we used to explain the technique. Waze may use similar techniques on Bluetooth or other proximity based communications when WiFi access is not available.

C. Graph-Based Sybil Detection

We apply graph-based Sybil detection algorithms to detect Sybils in Waze proximity graph. Graph-based Sybil detectors [33], [36]–[42] were originally proposed in social networks. They all rely on the key assumption that Sybils have difficulty to form edges with real users, which results in a sparse cut between the Sybil and non-Sybil regions in the social graph. Because of the limited number of "attack edges" between Sybils and non-Sybils, a random walk from non-Sybil region has a higher landing probability to land on a non-Sybil node than a Sybil node.

Although this assumption may not always hold in online social networks [43], it holds well for the proximity graph. In online social networks, Sybils may build "attack edges" by befriending with real users (*e.g.*, using attractive female photos) [43]. However, in a proximity graph, building an attack edge requires *physical* collocations. With the WiFi authentication, it's difficult to build attack edges using software simulations alone in a massive, automated manner (*e.g.*, for tens of thousands of Sybil devices). In addition, the authentication is done in the background without human involvement, which further eliminates the chance for Sybils to trick real users to add edges.

SybilRank: We choose SybilRank as our main algorithm. Compared to its counterparts [36]–[38], SybilRank achieves a higher accuracy at a lower computational cost, and has been successfully deployed in a real-world social network with tens of millions of users [39]. At the high-level, SybilRank ranks the nodes based on how likely they are Sybils. The algorithm starts with multiple trusted nodes in the graph. It iteratively computes the landing probability for short random walks (originated from trusted nodes) to land on all other nodes. The landing probability is normalized by the node's degree, which acts as the trust score for ranking. Intuitively, short random walks from trusted nodes are very unlikely to traverse the few attack edges to reach Sybil nodes, and thus Sybils' scores should be lower.

SybilRank is designed to rank Sybils and allows system administrators to go through the ranked list to decide which accounts to suspend. As shown in [39], in practice, the administrators may set a cut-off value for the trust score and label the tail of the list as Sybils. For example, administrators can go through the ranked list from the most suspicious accounts to the least suspicious ones. They can stop at some point (the cut-off value) when they find the non-Sybil rate gets too high.

The original SybilRank works on unweighted social graphs. We modified it to work on our weighted proximity graph: when a node propagates trust (or performs random walks) to its neighbors, instead of splitting the trust equally, it distributes proportionally based on the edge weights. This actually makes it harder for Sybils to evade SybilRank—they will need to build more high-weight attack edges to real users to receive trust.

SybilSCAR: In addition, we also consider a more recent algorithm SybilSCAR [41] for comparison purposes. SybilSCAR unifies multiple graph-based Sybil detection algorithms into a single framework and proposes a new set of rules for label propagating. However, SybilSCAR requires a small number of known Sybils as well as trusted nodes as seeds, and thus is not our first choice (SybilRank only needs a few trusted nodes). SybilSCAR iteratively propagates label information (Sybil and non-Sybil) from nodes to their neighbors. The underlying assumption is the homophily property of social graphs, *i.e.*, real users are more likely to connect with real users and Sybils are more likely to connect with Sybils, which is applicable to our proximity graph.

VII. Countermeasure Evaluation

We use simulations to evaluate the effectiveness of our proposed defense. We focus on evaluating the feasibility and cost for attackers to maintain a large number of Sybils after the Sybil detection is in place. We quantify the cost by the number of attack edges a Sybil must establish with real users. In practice, this translates into the effort taken to physically drive around and use physical devices (with WiFi radios) per Sybil to complete proximity authentication. In the following, we first describe our simulation setup, and then present the key findings and their implications on Waze.

A. Evaluation Setup

We first discuss how we construct a synthetic proximity graph for our evaluation, followed by the counter strategies taken by attackers to evade detection. Finally, we describe the evaluation metrics for Sybil detection.

Simulating Proximity Graphs: We use well-known models on human encountering to create synthetic proximity graphs. This is because, to the best of our knowledge, there is no public *per-user* mobility dataset with sufficient scale and temporal coverage to support our evaluation. Also, directly crawling large-scale, per-user mobility trace from Waze can lead to questionable privacy implications, and thus we exclude this option.

Existing literatures [44]–[48] all suggest that human (and vehicle) encounter patterns display strong scale-free and "small-world" properties [49]. Thus we follow the methodology of [44] to simulate a power-law based encounter process among Waze users. Given a user population N, we first assign each user an encounter probability following a power-law distribution (α =2 based on the empirical values [44], [50]). We then simulate user encounter over time, by adding edges to the graph based on the joint probability of the two nodes.

For our evaluation, we produce a proximity graph for $N = 10000$ normal users and use the snapshot when 99.9% of nodes are connected. Note that as the graph gets denser over time, it is harder for Sybils to blend into normal user regions. We use this graph to simulate the lower-bound performance of Sybil detection.[2]

[2]Validated by experiments: a denser, 99.99% connected graph can uniformly improve Sybil detection accuracy.

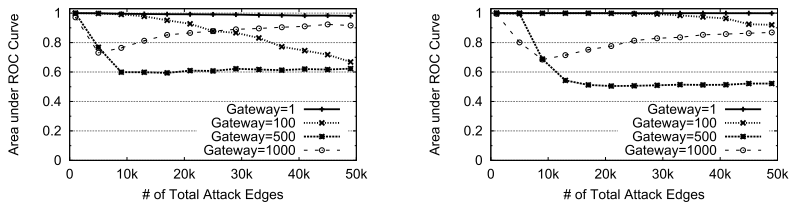


Fig. 8. SybilRank: AUC with respect to number of attack edges, where Sybils form power-law inner connections. (a) Sybil inner connection avg. degree = 5. (b) Sybil inner connection avg. degree = 10.

Note that by following a power-law encountering probability, our model already considers the effect of new users or inactive users. In this graph, only a small portion of active users has a high degree, while most users (including new users) have a low degree due to a low encountering probability. In practice, Waze can use their real graphs for this experiment.

Attacker Models: In the presence of Sybil detection, an attacker will try mixing their Sybils into the proximity graph.

We consider the following strategies:

1) **Single-Gateway** – An attacker first takes one Sybil account (as the gateway) to build attack edges to normal users. Then the attacker connects the remaining Sybils to this gateway. In practice, this means the attacker only needs to take one physical phone to go out and encounter normal users.
2) **Multi-Gateways** – An attacker distributes the attack edges to multiple gateways, and then evenly spreads the other Sybils across the gateways. This helps the Sybils to blend in with normal users. The attacker pays an extra cost in terms of using multiple real devices to build attack edges.

The attacker also builds edges among its own Sybils to maintain a legitimate degree distribution, and boost each other's trust score. In our simulation, we follow the scale-free distribution to add edges among Sybils mimicking normal user region (we did not use a fully connected network between Sybils since it is more easily detectable).

Evaluation Metrics: To evaluate Sybil detection efficacy, we use the standard false positive (negative) rate, and the Area under the Receiver Operating Characteristic curve (AUC) used by SybilRank [39]. AUC represents the probability that SybilRank ranks a random Sybil node lower than a random non-Sybil node. Its value ranges from 0 to 1, where 1 means the ranking is perfect (all Sybils are ranked lower than non-Sybils), 0 means the ranking is always flipped, and 0.5 matches the result of random guessing. Compared to false positive (negative) rates, AUC is independent of the cutoff threshold, and thus comparable across experiment settings.

B. Results

Our evaluation primarily focuses on SybilRank, and we briefly discuss the results of SybilSCAR in the end.

Accuracy of Sybil Detection: We assume the attacker seeks to embed 1000 Sybils into the proximity graph. We use either single- or multi-gateway approaches to build attack edges on the proximity graph by connecting Sybils to randomly chosen normal users. We then add edges between Sybil nodes, following the power-law distribution and producing an average weighted degree of either 5 or 10 (to emulate different Sybil subgraph density). We randomly select 10 trusted nodes to bootstrap trust for SybilRank and run it on the proximity graph. We repeat each experiment 50 times.

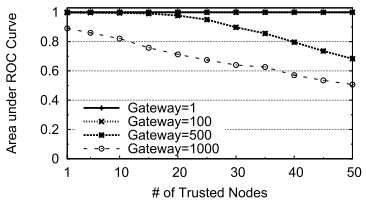


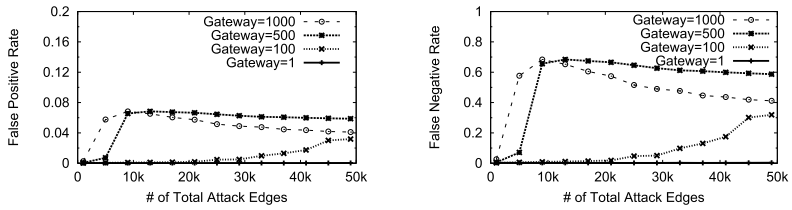
Fig. 9. SybilRank: Impact of # of trusted nodes (average degree =10 for Sybil region; 5K attack edges).

Figure 8 shows that the Sybil detection mechanism is highly effective. For attackers of the single-gateway model, the AUC is very close to 1 (> 0.983), indicating Waze can identify almost all Sybils even after the attacker established a large number of attack edges, *e.g.*, 50000. Meanwhile, the multi-gateway method helps attackers add "undetected" Sybils, but the number of gateways required is significant. For example, to maintain 1000 Sybils, *i.e.*, by bringing down AUC to 0.5, the attacker needs at least 500 as gateways. In practice, this means wardriving with 500+ physical devices to meet real users, which is a significant overhead.

Interestingly, the 1000-gateway result (where every Sybil is a gateway) shows that, at certain point, adding more attack edges can actually hurt Sybils. This is potentially due to the fact that SybilRank uses node degree to normalize trust score. For gateways that connect to both normal users and other Sybils, the additional "trust" received by adding more attack edges cannot compensate the penalty of degree normalization.

For a better look at the *detection accuracy*, we convert the AUC in Figure 8(b) to false positives (classifying real users as Sybils) and false negatives (classifying Sybils as real users). For simplicity, we set a cutoff value to mark the bottom 10% of the ranked nodes as Sybils. This cutoff value is only to convert the error rate. In practice, Waze can optimize this value based on the trust score or manual examination. As shown in Figure 10, SybilRank is highly accurate to detect Sybils when the number of gateways is less than 100. Again, 100 gateways incur high cost in practice.

Next we quickly examine the impact of trusted nodes to Sybil detection. Figure 9 shows a small number of trusted node is enough to run SybilRank. Interestingly, adding more trusted nodes can slightly hurt Sybil detection, possibly because it gives the attacker (gateways) a higher chance to receive trust. In practice, multiple trusted nodes can help SybilRank



(a) (b)

Fig. 10. SybilRank: Detection error rates with respect to number of attack edges. We set average degree =10 for Sybils' power-law inner connections. (a) False Positive Rate. (b) False Negative Rate.

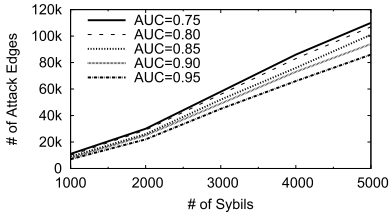


Fig. 11. SybilRank: # of attack edges needed to maintain x Sybil devices with respect to different AUC level.

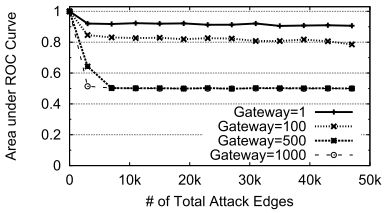


Fig. 12. SybilSCAR: # of attack edges vs. AUC (verage degree = 5 for Sybil region).

overcome potential community structures in proximity graph (*e.g.*, users of the same city form a cluster). So Waze should place trusted nodes accordingly to cover geographic clusters.

Cost of Sybil Attacks: Next, we infer the rough cost of attackers on implementing successful Sybil attacks. For this we look at the number of attack edges required to successfully embed a given number of Sybils. Our experiment assumes the attacker uses 500 gateways and builds power-law distributed inner connections with average degree = 10. Figure 11 shows the number of attack edges required to achieve a specific AUC under SybilRank as a function of the target number of Sybils. We see that the attack edge count increases linearly with the Sybil count. The cost of Sybil attack is high: to maintain 3000 Sybils, the attacker must make 60,000 attack edges to keep AUC below 0.75, and spread these attack edges across 500 high-cost gateways.

Smaller Sybil Groups: We briefly examine how effective our system is in detecting much smaller Sybil groups. We test Sybil groups with size of 20, 50 and 100 using a single-gateway approach. We configure 50K attacking edges for Sybils with inner degree = 10. The resulting AUC of Sybil detection is 0.90, 0.95 and 0.99 respectively. This confirms our system can effectively identify small Sybil groups as well.

Handling False Positives: For the few false positives (*e.g.*, new accounts without an edge yet), Waze can handle them properly without affecting much of its functionality. For example, Waze can apply "temporary" restrictions, by lowering their weights in traffic aggregation, and enforcing strict rate limits for querying nearby users. Once the new accounts establish some edges after one trip, Waze then can release the restriction.

SybilSCAR Results: We perform a quick evaluation on SybilSCAR. We set the average degree of the Sybil region as 5, and feed 10 random trusted nodes and 10 random known Sybils to bootstrap SybilSCAR. The results are shown in Figure 12. SybilSCAR performs well under the single-gateway setting (AUC above 0.9). The AUC still remains above 0.8 under 100 gateways. The results suggest that our system is not too sensitive to the choice of the Sybil detection algorithm. Once the proximity graph is constructed, existing off-the-shelf Sybil detection algorithms can help to support the system.

VIII. Our Interactions With Waze

After our study, we have taken active steps to inform Google/Waze team of our results and helped them to mitigate the threat. In this section, we briefly describe our interactions with Waze team and their new security measures.

Informing Waze Team Directly: Before the first writeup of our work in November 2014, we sought to inform the Google Waze team of our findings. We first used multiple existing Google contacts on the Security and Android teams to reach out to Waze. When that failed, we got in touch with Niels Provos, who relayed information about our project to the Waze team.

As of October 2015, we observed a major change in Waze app on how the app reports user GPS to the server (and other users). In the new version, the app only reports user GPS when the user is actively driving (moving at a moderate/fast rate of speed). In addition, Waze automatically shuts down if the user puts it in the background, and has not driven for a while. To resume GPS reporting, users must manually bring the app to the foreground. Finally, Waze hides users' starting and destination locations of their trips. While online documentation claims that these optimizations are to reduce energy usage for the app, we are gratified by the dramatic steps taken to limit user tracking and improve user privacy. These changes indeed reduce the amount of GPS data (by nearly a factor of 10x) sent to the server and made available to potential attackers through the APIs.

Informing Waze Through News Media: After the above updates, attackers could still track users who are actively using the app. To further raise the awareness of the attack, we pitched our work to Fusion (a major media outlet). On April 26, 2016, Fusion covered our story, which went viral within 24 hours with 20+ followup reports from news media all around the

world. This time, Waze immediately issued a response on the next day [51] and a series of updates to the app. First, Waze disabled the social feature in older versions (v3.8 or lower). In addition, the new app uses special encoding on the communication APIs so that the API parameters are no longer human-readable. However, after some quick analysis, we found the encoding was implemented with Google Protocol Buffer. Based on standard format of the parameter values, we managed to crack the encoding and extracted the new APIs within a day. We validated that our attack still worked, and informed Waze of our finding.

Working With Waze: As of May 2016, the product manager of Waze reached out to us to start a collaboration to improve Waze security. Since then, Waze started to require a two-factor authentication through SMS before showing any identifiable information to nearby users. More importantly, we strongly suggested Waze removing the globally unique identifiers (account creation time) and usernames. Waze followed our suggestion and now it is very difficult to persistently track users over multiple trips.

To assess the effectiveness of the SMS-based verification, we tested to bypass this using temporal SMS services [21]. Our attempt succeeded. Once the account got verified, our Sybil device can then communicate with Waze server to track users. We reported our findings and also pointed them to our proximity graph based defense (§VI). It is an on-going effort to further raise the bar for attackers.

Thus far, our efforts have led to significant improvement to the security and privacy in Waze. After the back-and-forth interaction, much less amount of location information is shared about users. Currently, only active users (who are driving on the road with Waze app on the foreground) can be tracked. In addition, we convinced Waze to remove the globally unique identifiers of users, making it very difficult to track users across multiple trips.

IX. Broader Implications

While our experiments and defenses have focused strictly on Waze, our results are applicable to a wider range of mobile applications that rely on geolocation for user-contributed content and metadata. Examples include location based check-in services (Foursquare, Yelp), mobile navigation systems (Waze, Moovit), crowdsourced taxi services (Uber, Lyft), mobile dating apps (Tinder, Bumble), anonymous mobile communities (Yik Yak, Whisper) and location-based gaming apps (Pokemon Go).

These systems face two common challenges exposing them to potential attacks. First, our efforts show that it is difficult for app developers to build a truly secure channel between the app and the server. There are numerous avenues for an attacker to reverse-engineer and mimic an app's API calls, thereby creating "cheap" virtual devices and launching Sybil attack [4]. Second, there are no deployed mechanisms to authenticate location data (*e.g.*, GPS report). Without a secure channel to the server and authenticated location, these mobile apps are vulnerable to automated attacks ranging from nuisance (prank calls to Uber) to malicious content attacks (large-scale rating manipulation on Yelp).

A. Attacking Other Apps

To validate our point, we run a quick empirical analysis on a broad class of mobile apps to understand how easy it is to reverse-engineer their APIs and inject falsified data into the system. We pick one app from each category including Foursquare, Uber, Tinder, Yik Yak and Pokemon Go (an incomplete list). We find that, although all the listed apps use TLS/SSL to encrypt their network traffic, their APIs can be fully exposed by the method in §IV. For each app, we were able to build a light-weight client using python script, and feed arbitrary GPS to their key function calls. For example, with forged GPS, a group of Foursquare clients can deliver large volumes of check-ins to a given venue without physically visiting it; On Uber, one can distribute many virtual devices as sensors, and passively monitor and track all drivers within a large area (see §V). Similarly for Yik Yak and Tinder, the virtual devices make it possible to perform wardriving in a given location area to post and collect anonymous Yik Yak messages or Tinder profiles. In addition, apps like Tinder also display the geographical distance to a nearby user (*e.g.*, 1 mile). Attacker can use multiple virtual devices to measure the distance to the target user, and "triangulate" that user's exact location [52]. Finally, for Pokemon Go, we can use simulated devices to capture pokemons without physically walking outside like other players do (cheating in the game).

B. New Countermeasures in the Wild

After our initial report was published, we have observed new countermeasures from these apps. For example, Yik Yak uses HMAC (keyed-hash message authentication code) to authenticate their APIs. The app embeds a key in the binary to generate authentication code. Any API calls without the code are not accepted. In this case, the attacker will need to extract the key from binary to build a Sybil device. In addition, apps like Twitter and Periscope have adopted SSL pinning to spot self-signed certificates. Attacker will need to replace the pinned certificate in order to set up the HTTPS proxy to inspect API calls. Further research is needed to empirically understand the usage and effectiveness of different countermeasures in the wild.

X. Related Work

Security in Location-Based Services: Location-based services face various threats, ranging from rogue users reporting fake GPS [2], [53], to malicious parties compromising user privacy [54]. A related study on Waze [55] demonstrated that small-scale attacks can create traffic jams or track user icons, with up to 15 mobile emulators. Our work differs in two key aspects. First, we show that it's possible to reverse engineer its APIs, enabling light-weight Sybil devices (simple scripts) to replace full-stack emulators. This increase the scale of potential attacks by orders of magnitude, to thousands of Waze clients per commodity laptop. The impact of thousands of virtual vehicles is qualitatively different from 10-15 mobile simulators. Second, as possible defenses, [55] cites known tools such as phone number/IP verification, or location authentication with cellular towers, which have limited applicability

(see §6). In contrast, we propose a novel proximity graph approach to detect and constrain the impact of virtual devices.

Researchers have proposed to preserve user location privacy against map services such as Waze and Google. Earlier studies apply location cloaking by adding noise to the GPS reports [56]. Recent work use zero-knowledge [57] and differential privacy [58] to preserve the location privacy of individual users. Our work differs by focusing on the attacks against the map services.

Mobile Location Authentication: Defending against forged GPS is challenging. One direction is to authenticate user locations using wireless infrastructures: WiFi APs [29], [30], cellular base stations [29], [30] and femtocells [59]. Devices must come into physical proximity to these infrastructures to be authenticated. But it requires cooperation among a wide range of infrastructures (also modifications to their software/hardware), which is impractical for large-scale services like Waze. Our work only uses a small number of trusted infrastructures to bootstrap, and relies on peer-based trust propagation to achieve coverage. Other researchers have proposed "peer-based" methods to authenticate collocated mobile devices [34], [60], [61].

Different from existing work, we use peer-based collocation authentication to build proximity graphs for Sybil detection, instead of directly authenticating a device's physical location.

XI. Conclusion

We describe our efforts to identify and study a range of attacks on crowdsourced map services. We identify a range of single and multi-user attacks, and describe techniques to build and control groups of virtual vehicles (ghost riders) to amplify these attacks. Our work shows that today's mapping services are highly vulnerable to software agents controlled by malicious users, and both the stability of these services and the privacy of millions of users are at stake. While our study and experiments focus on the Waze system, we believe the large majority of our results can be generalized to crowdsourced apps as a group. We propose and validate a suite of techniques that help services build proximity graphs and use them to effectively detect Sybil devices. Throughout this work, we have taken active steps to isolate our experiments and prevent any negative consequence on real Waze users. We also proactively informed Waze team of theses attacks, and worked with them to mitigate the threat.

References

[1] N. Stefanovitch, A. Alshamsi, M. Cebrian, and I. Rahwan, "Error and attack tolerance of collective problem solving: The DARPA shredder challenge," *EPJ Data Sci.*, vol. 3, no. 1, pp. 1–27, 2014.

[2] B. Carbunar and R. Potharaju, "You unlocked the Mt. Everest badge on Foursquare! Countering location fraud in geosocial networks," in *Proc. MASS*, 2012, pp. 182–190.

[3] Z. Zhang *et al.*, "On the validity of geosocial mobility traces," in *Proc. HotNets*, 2013, p. 11.

[4] J. R. Douceur, "The Sybil attack," in *Proc. IPTPS*, 2002, pp. 251–260.

[5] S. Cheng, *Uber's Terrifying 'Ghost Drivers' are Freaking out Passengers in China*. New York, NY, USA: Quartz, Sep. 2016.

[6] Y. Wang, "Ghost drivers are just one of Uber China's problems following DIDI takeover," *Forbes*, Sep. 2016.

[7] M. Wehner, "How to cheat at Pokémon Go and catch any Pokemon you want without leaving your couch," *DailyDot*, Jul. 2016.

[8] *How to Avoid Getting Banned in Pokemon Go While Location Spoofing*, Cydiageeks, San Francisco, CA, USA, Jul. 2016.

[9] V. Goel, "Maps that live and breathe with data," The New York Times, New York, NY, USA, Tech. Rep., Jun. 2013. [Online]. Available: https://www.nytimes.com/2013/06/11/technology/mobile-companies-crave-maps-that-live-and-breathe.html

[10] *Google Maps and Waze, Outsmarting Traffic Together*, Google Official Blog, Google, Mountain View, CA, USA, Jun. 2013.

[11] *GenyMotion Emulator*. Accessed: Jun. 2016. [Online]. Available: http://www.genymotion.com

[12] *Monkeyrunner*. Accessed: Jun. 2016. [Online]. Available: https://developer.android.com/studio/test/monkeyrunner/index.html

[13] B. Reed, "Google Maps becomes Google's second 1 billion-download hit," *Yahoo! News*, Jun. 2014.

[14] *Charles Proxy*. Accessed: Jun. 2016. [Online]. Available: http://www.charlesproxy.com

[15] D. Sounthiraraj, J. Sahs, G. Greenwood, Z. Lin, and L. Khan, "SMV-HUNTER: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in Android Apps," in *Proc. NDSS*, 2014. [Online]. Available: http://dx.doi.org/10.14722/ndss.2014.23205

[16] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations," in *Proc. IEEE S&P*, May 2014, pp. 114–129.

[17] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking SSL development in an appified world," in *Proc. CCS*, 2013, pp. 49–60.

[18] J. Osborne and A. Diquet, "Principal security engineer," iSEC Partners, San Francisco, CA, USA, Tech. Rep., 2012. [Online]. Available: https://media.blackhat.com/bh-us-12/Turbo/Diquet/BH_US_12_Diqut_Osborne_Mobile_Certificate_Pinning_Slides.pdf

[19] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of Android application security," in *Proc. USENIX Secur.*, 2011, p. 2.

[20] K. Thomas *et al.*, "Framing dependencies introduced by underground commoditization," in *Proc. WEIS*, 2015. [Online]. Available: http://www.econinfosec.org/archive/weis2015/papers/WEIS_2015_thomas.pdf

[21] K. Thomas *et al.*, "Dialing back abuse on phone verified accounts," in *Proc. CCS*, 2014, pp. 465–476.

[22] M. Motoyama *et al.*, "Re: CAPTCHAs-understanding CAPTCHA-solving services in an economic context," in *Proc. USENIX Secur.*, 2010, p. 3.

[23] *IMEI Database*. Accessed: Jun. 2016. [Online]. Available: http://www.imei.info/phonedatabase/

[24] *IMEI Generator*. Accessed: Jun. 2016. [Online]. Available: https://www.getnewidentity.com/imei-generator.php

[25] A. Das, N. Borisov, and M. Caesar, "Tracking mobile Web users through motion sensors: Attacks and defenses," in *Proc. NDSS*, 2016, pp. 1–15.

[26] A. Das, N. Borisov, E. Chou, and M. H. Mughees, "Smartphone fingerprinting via motion sensors: Analyzing feasibility at large-scale and studying real usage patterns," *CoRR*, vol. abs/1605.08763, May 2016. [Online]. Available: http://arxiv.org/abs/1605.08763

[27] *About Waze: Privacy*. Accessed: Jun. 2016. [Online]. Available: https://support.google.com/waze/answer/6071193?hl=en

[28] *Open Source Routing Machine (OSRM)*. Accessed: Jun. 2016. [Online]. Available: http://map.project-osrm.org.

[29] W. Luo and U. Hengartner, "Proving your location without giving up your privacy," in *Proc. HotMobile*, 2010, pp. 7–12.

[30] S. Saroiu and A. Wolman, "Enabling new mobile applications with location proofs," in *Proc. HotMobile*, 2009, p. 3.

[31] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun, "Smartphones as practical and secure location verification tokens for payments," in *Proc. NDSS*, 2014, pp. 1–19.

[32] S. Saroiu and A. Wolman, "I am a sensor, and I approve this message," in *Proc. HotMobile*, 2010, pp. 37–42.

[33] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove, "An analysis of social network-based Sybil defenses," *SIGCOMM*, vol. 40, no. 4, pp. 363–374, 2010.

[34] Z. Zhu and G. Cao, "Toward privacy preserving and collusion resistance in a location proof updating system," *IEEE Trans. Mobile Comput.*, vol. 12, no. 1, pp. 51–64, Jan. 2013.

[35] J. M. Tjensvold. (2007). *Comparison of the IEEE 802.11, 802.15.1,802.15.4 and 802.15.6 Wireless Standards*. [Online]. Available: http://janmagnet.files.wordpress.com/2008/07/comparison-ieee-802-standards.pdf

[36] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "SybilGuard: Defending against Sybil attacks via social networks," in *Proc. SIGCOMM*, 2006, pp. 267–278.

[37] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "SybilLimit: A near-optimal social network defense against Sybil attacks," in *Proc. IEEE S&P*, May 2008, pp. 3–17.

[38] G. Danezis and P. Mittal, "Sybilinfer: Detecting Sybil nodes using social networks," in *Proc. NDSS*, 2009, pp. 1–15.

[39] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro, "Aiding the detection of fake accounts in large scale social online services," in *Proc. NSDI*, 2012, p. 15.
[40] S. Misra, A. S. M. Tayeen, and W. Xu, "SybilExposer: An effective scheme to detect Sybil communities in online social networks," in *Proc. ICC*, 2016, pp. 1–6.
[41] B. Wang, L. Zhang, and N. Z. Gong, "SybilSCAR: Sybil detection in online social networks via local rule based propagation," in *Proc. INFOCOM*, 2017, pp. 1–9.
[42] J. Jia, B. Wang, and N. Z. Gong, "Random walk based fake account detection in online social networks," in *Proc. DSN*, 2017, pp. 273–284.
[43] Z. Yang *et al.*, "Uncovering social network sybils in the wild," in *Proc. IMC*, 2011. p. 2.
[44] A. G. Miklas *et al.*, "Exploiting social interactions in mobile systems," in *Proc. Ubicomp*, 2007, pp. 409–428.
[45] F. Cunha, A. C. Viana, R. A. F. Mini, and A. A. F. Loureiro, "Is it possible to find social properties in vehicular networks?" in *Proc. ISCC*, 2014, pp. 1–6.
[46] F. Tan, Y. Borghol, and S. Ardon, "EMO: A statistical encounter-based mobility model for simulating delay tolerant networks," in *Proc. WOWMOM*, 2008, pp. 1–8.
[47] T. Hossmann, T. Spyropoulos, and F. Legendre, "Know thy neighbor: Towards optimal mapping of contacts to social graphs for DTN routing," in *Proc. INFOCOM*, 2010, pp. 1–9.
[48] X. Liu *et al.*, "Exploring social properties in vehicular ad hoc networks," in *Proc. Internetware*, 2012, p. 24.
[49] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
[50] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Rev.*, vol. 51, no. 4, pp. 661–703, 2009.
[51] *Waze's Response to Our Research*. Accessed: Apr. 2016. [Online]. Available: https://blog.waze.com/2016/04/privacy-and-waze.html
[52] G. Wang *et al.*, "Whispers in the dark: Analysis of an anonymous social network," in *Proc. IMC*, 2014, pp. 137–150.
[53] W. He, X. Liu, and M. Ren, "Location cheating: A security challenge to location-based social network services," in *Proc. ICDCS*, 2011, pp. 740–749.
[54] Y.-A. de Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, "Unique in the crowd: The privacy bounds of human mobility," *Sci. Rep.*, vol. 3, Mar. 2013, Art. no. 1376.
[55] M. B. Sinai, N. Partush, S. Yadid, and E. Yahav, "Exploiting social navigation," *Black Hat Asia*, Oct. 2015. [Online]. Available: http://www.blackhat.com/docs/asia-15/materials/asia-15-Partush-Exploiting-Social-Navigation.pdf
[56] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proc. MobiSys*, 2003, pp. 33–42.
[57] T. Jeske, "Floating car data from smartphones: What Google and Waze know about you and how hackers can control traffic," in *Proc. Black Hat*, 2013, pp. 1–12.
[58] J. W. S. Brown, O. Ohrimenko, and R. Tamassia, "Haze: Privacy-preserving real-time traffic statistics," in *Proc. SIGSPATIAL*, 2013, pp. 540–543.
[59] J. Brassil, P. K. Manadhata, and R. Netravali, "Traffic signature-based mobile device location authentication," *IEEE Trans. Mobile Comput.*, vol. 13, no. 9, pp. 2156–2169, Sep. 2014.
[60] J. Manweiler, R. Scudellari, and L. P. Cox, "SMILE: Encounter-based trust for mobile social services," in *Proc. CCS*, 2009, pp. 246–255.
[61] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing," in *Proc. NDSS*, 2011, pp. 1–12.

Gang Wang received the B.E. degree in electrical engineering from Tsinghua University, Beijing, China, in 2010, and the Ph.D. degree in computer science from the University of California (UC) at Santa Barbara, CA, USA, in 2016. He is currently an Assistant Professor with the Department of Computer Science, Virginia Tech, VA, USA. His research interests include security and privacy, online social networks, mobile networks, and crowdsourcing. He was a recipient of the Google Faculty Research Award in 2018, the Best Practical Paper Award from ACM SIGMETRICS in 2013, and the Outstanding Dissertation Award in 2016 and Ph.D. Dissertation Fellowship in 2015 from UC Santa Barbara.

Bolun Wang received the B.S. degree in electrical engineering from Tsinghua University in 2009. He is currently pursuing the Ph.D. degree with the Computer Science Department, University of California at Santa Barbara, Santa Barbara. He is currently with The University of Chicago as a Visiting Scholar. He published papers in ACM TWEB, ACM IMC, MobiSys, and CSCW. His research interests are security and privacy of mobile applications and online systems.

Tianyi Wang received the B.S. and Ph.D. degrees from the Department of Electronic Engineering, Tsinghua University, in 2011 and 2016, respectively. He was a Research Scientist at Baidu Research. He is currently holds a senior research and development position at ByteDance Inc. He published over 10 referred papers in international journals and conferences, including ACM TWEB, *IEEE Communications Magazine*, USENIX Security, ACM IMC, MobiSys, and CSCW. His research interests include data mining, machine learning, NLP, and security, mostly from a data-driven perspective.

Ana Nika received the B.Sc. degree in computer science and the M.Sc. degree in communication systems and networks from the National and Kapodistrian University of Athens, Greece, in 2011 and 2007, respectively, and the Ph.D. degree in computer science from the University of California at Santa Barbara, Santa Barbara, in 2017. She is currently a Software Engineer at Microsoft.

Haitao (Heather) Zheng (F'15) received the Ph.D. degree from the University of Maryland, College Park, MD, USA, in 1999. After spending six years as a Researcher in industry labs (Bell-Labs, USA, and Microsoft Research Asia), she joined the Faculty of the University of California at Santa Barbara in 2005, and moved to The University of Chicago in 2017. She is currently the Neubauer Professor of computer science with The University of Chicago, where she also co-directs the SANDLab with a broad research coverage on wireless networking and systems, mobile computing, security, and data mining and modeling. Her research has been featured by a number of media outlets, such as *The New York Times, Boston Globe, LA Times, MIT Technology Review*, and *Computer World*. She has received a number of awards, such as the *MIT Technology Review's* TR-35 Award (Young Innovators Under 35) and the World Technology Network Fellow Award. She recently served as the TPC Co-Chair of MobiCom'15 and DySPAN'11. She is currently serving on the steering committees of MobiCom.

Ben Y. Zhao received the B.S. degree from Yale University in 1997, and the Ph.D. degree from the University of California at Berkeley in 2004. He is currently the Neubauer Professor of computer science with The University of Chicago. He has published over 150 publications in areas of security and privacy, networked systems, wireless networks, data-mining, and HCI (H-index 58). His work has been covered by media outlets, such as *Scientific American, The New York Times, Boston Globe, LA Times, MIT Tech Review*, and *Slashdot*. He was a recipient of the NSF CAREER Award, the *MIT Technology Review's* TR-35 Award (Young Innovators Under 35), the *ComputerWorld* Magazine's Top 40 Tech Innovators Award, the Google Faculty Award, and the IEEE ITC Early Career Award. He recently served as the TPC Co-Chair for the World Wide Web Conference (WWW 2016) and the ACM Internet Measurement Conference (IMC 2018). He is an ACM Distinguished Scientist.